第二章

遠端呈現與操控系統

在討論所提出遠端呈現與操控系統前，我們先來談一談在實現機器人的遠端遙控系統時所面臨的諸多挑戰[12]，下面列出主要的項目：

- 發展逼近真實的遠端呈現，包括視覺、聽覺和力覺等；
- 設計虛擬實境之輸入與輸出裝置設備，例如:頭盔式顯示器、立體眼鏡、力回饋搖桿和資料手套
- 改良操作裝置和遠端受控裝置之間的不相容性；
- 減少遠端遙控操作透過網路傳輸所造成的時間延遲效應；
- 提供輔助工具或輔助策略，協助使用者操控更有效率；
- 研發遠端智慧型控制器，以處理受控裝置與遠端環境互動時的問題；
- 加強人類和智慧型機器人之間的協調合作與互動。

本實驗室也針對這幾項挑戰進行研發，在對抗時間延遲的效應上，則利用雙向控制策略來處理，讓執行遠端遙控系統時可以穩定地完成任務；在互動方面，則發展出一智慧型控制器來處理力資訊，讓機器人碰觸到物體表面時可以維持一穩定的接觸力。基於之前的成果，爲了將遠端搖控機器人順利應用在安全監控

上，本篇論文主要著力於下列幾點：

1、 建構多視角的輔助視窗：由於電腦螢幕呈現觀看視角是有限的，無法像人類眼睛的視角那麼廣大，使得操控上容易發生盲點，爲了克服這問題，我們規劃多個視角，讓使用者可以從多個角度觀看虛擬場景，以利操控自動車執行巡邏任務。

2、 發展易於操控的輔助策略：自動車在不確定的變動環境下移動巡邏或執行工作時，可能會遇到多個障礙物，增加操控的困難程度，爲了讓使用者在操控時更有效率，所以發展出此策略，讓使用者只需選取目的地，路徑規畫演算法就會自動引導自動車避開障礙物並到達目的地。

3、 規劃提供障礙物資訊的回饋力：我們藉由力回饋搖桿傳回的力資訊作爲力覺回饋，同時利用虛擬實境作爲視覺回饋，使當使用者操控自動車靠近事先預定的限制地區或障礙物時，可以由回饋而來的視覺資訊得知障礙物的距離與方位，但由於觀察視角上容易因限制而發生障礙物距離上的誤判，儘管我們提供多個視角的切換，切換上的時間仍然讓資訊不夠即時，所以設計一力回饋操控系統，將模擬產生的虛擬力回傳到搖桿上，讓使用者因此感覺到阻力而得知障礙物的距離與方位，使其不能更進一步地驅使自動車逼近受限制地區或障礙物，同時也會從虛擬實境系統獲得視覺的回授訊號，加強遠端控制現場之臨場感。

2.1 系統架構

由於人類對於視覺、力覺這二種知覺較爲敏感，所以在遠端遙控系統的發展中，就較著重在這二方面的知覺呈現，因爲操控者可以透過這二種資訊，大致了解遠端的受控裝置和環境兩者之間發生互動時的情形；圖 2.1 爲一個典型的虛

擬實境網路機器人遠端操控系統示意圖，這系統主要由三個部分所組成的，包括虛擬實境的輸入/輸出裝置 (VR I/O devices)，還有虛擬實境引擎(VR engine)，以及伺服端的機器人和感測器，下面再針對這三部份做一下說明。

在虛擬實境的輸入/輸出裝置 (VR I/O devices) 部分，像是立體眼鏡、立體環繞音響以及力回饋搖桿，這些提供了互動的人機介面裝置，透過這些裝置，操縱者可以輸入命令到電腦中再去控制遠端的機器人，同時操縱者也可以感受到知覺回饋的感覺，在視覺的呈現上，可以讓人彷如置身在遠端環境；力覺的呈現方面，則可以讓人感覺是自己親自與遠端環境做互動時手的力覺；第二部分是虛擬實境引擎 (VR engine)，這部分是整個系統的的核心，它規劃虛擬環境的模型，以及繪出場景和管理物體在模擬迴圈中的行為，另外它也將從真實世界中感測器所量測的資料傳輸給使用者，以及發送命令給遠端的機器人，作為虛擬實境和真實世界兩者之間溝通的橋樑，因此實現虛擬實境引擎時最重視就是電腦效能，它的好壞影響到整個遙控機器人系統。第三部份伺服端的機器人和感測器，這大致上包含了遠端機器人、驅動裝置 (Drive units) 及感測器，在這部分首先驅動裝置會先接收由近端所傳送過來的命令，接著去控制機器人的行為，感測器再將所量測的資訊傳回近端給使用者，像是力、位置或力矩等資訊。

基於圖 2.1 的架構，在我們的實驗室裡，已經發展出一遠端遙控機器人系統，如圖 2.2，此系統採用分散式的架構概念，使用了兩台電腦，一台放在主控端，負責提供視覺和溝通力回饋搖桿，另一台放在伺服端，除了傳送資訊和接收命令外，另外也負責傳送透過遠端 CCD 攝影機所獲取的影像資訊，藉由此資訊計算出物體在影像中的位置，並找出物體和機器人之間的實際空間關係，有了物

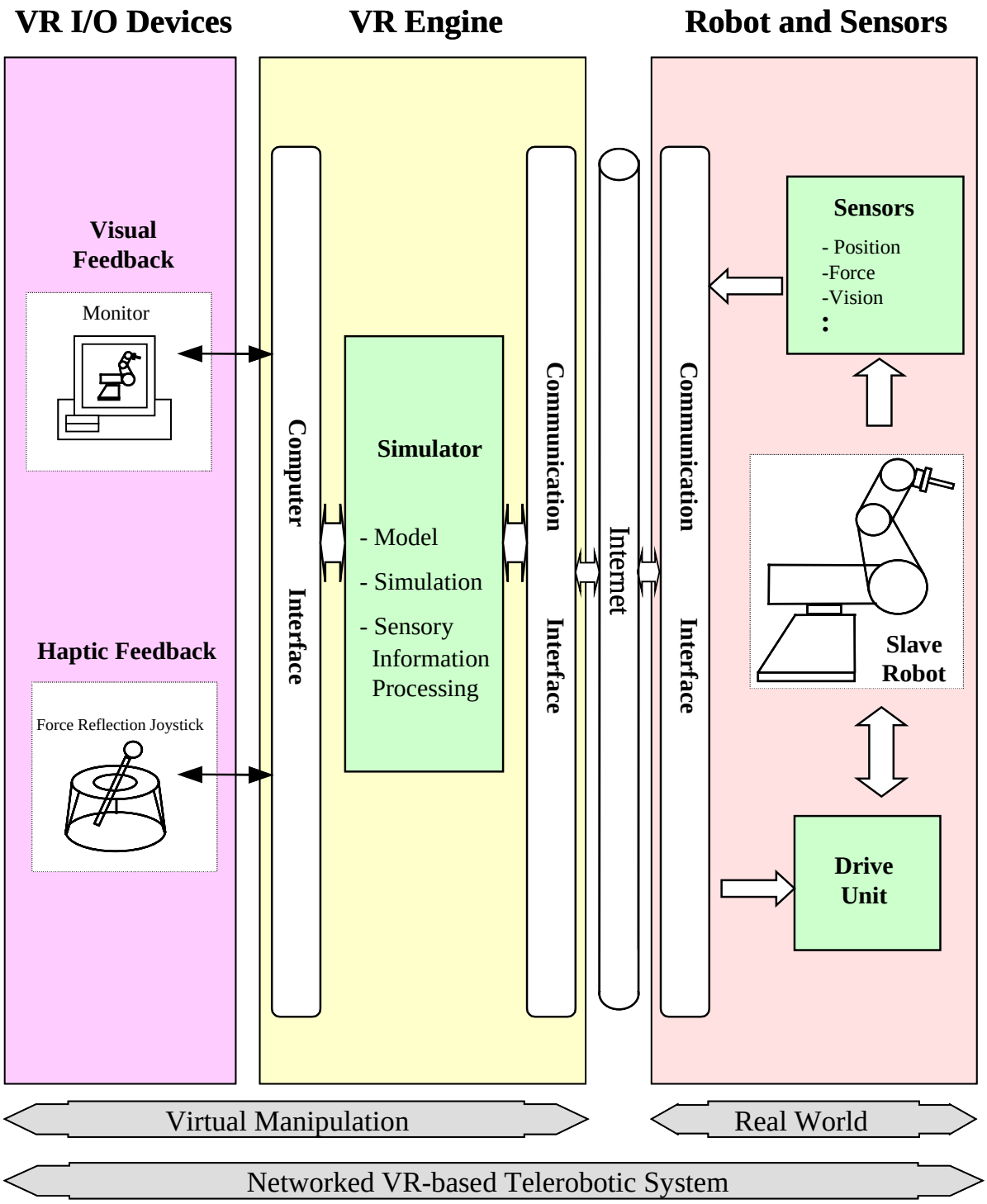


圖 2.1 典型的虛擬實境網路機器人遠端操控系統示意圖

體的位置資訊，即可即時更新及校正虛擬場景，使其能自動同步地獲得與真實環境的一致性[18]；而感測器則可以量測出物體的物理參數，像是質量、黏滯係數及彈性係數，虛擬實境就可以利用這些參數，將物體繪製出更精確的模型，並且可以提供更逼真的力覺給使用者。

將上述遠端遙控技術引進安全巡邏自動車系統中，讓使用者可以在任何地方進行遙控操控的任務，本論文是針對安全巡邏自動車系統來發展出虛擬實境安全巡邏自動車遠端呈現與操控系統，其系統架構如圖 2.3 所示。此系統是位於操作者端，主要包括使用者、力回饋搖桿操控系統、以及智慧型人機介面系統；使用者經由觀看遠端呈現系統所顯示的虛擬場景，並利用力回饋搖桿操控系統送出操控指令，再透過人機介面調節處理後，傳到遠端呈現系統來移動虛擬場景中的自動車，當自動車與場景中的設備或是物體快要發生碰撞時，則利用平面與探針(Plane and probe)方法[15]產生所相對應回饋的力，此力透過人機介面處理，經由力回饋操控器，再帶給使用者手中的反應力，讓使用者得到障礙物的訊息。同時，使用者可以藉由智慧型人機介面系統，以滑鼠在虛擬實境上的游標指示自動車該達到的位置。根據這個位置資訊，我們可以利用路徑規畫來產生最佳路徑，同時達到避免碰撞及增加效率的目的，而所規畫出來的最佳路徑，經由一些計算，我們可以產生出導引自動車追蹤此路徑的控制命令，例如前進、後退、轉向等，來移動虛擬場景的自動車。

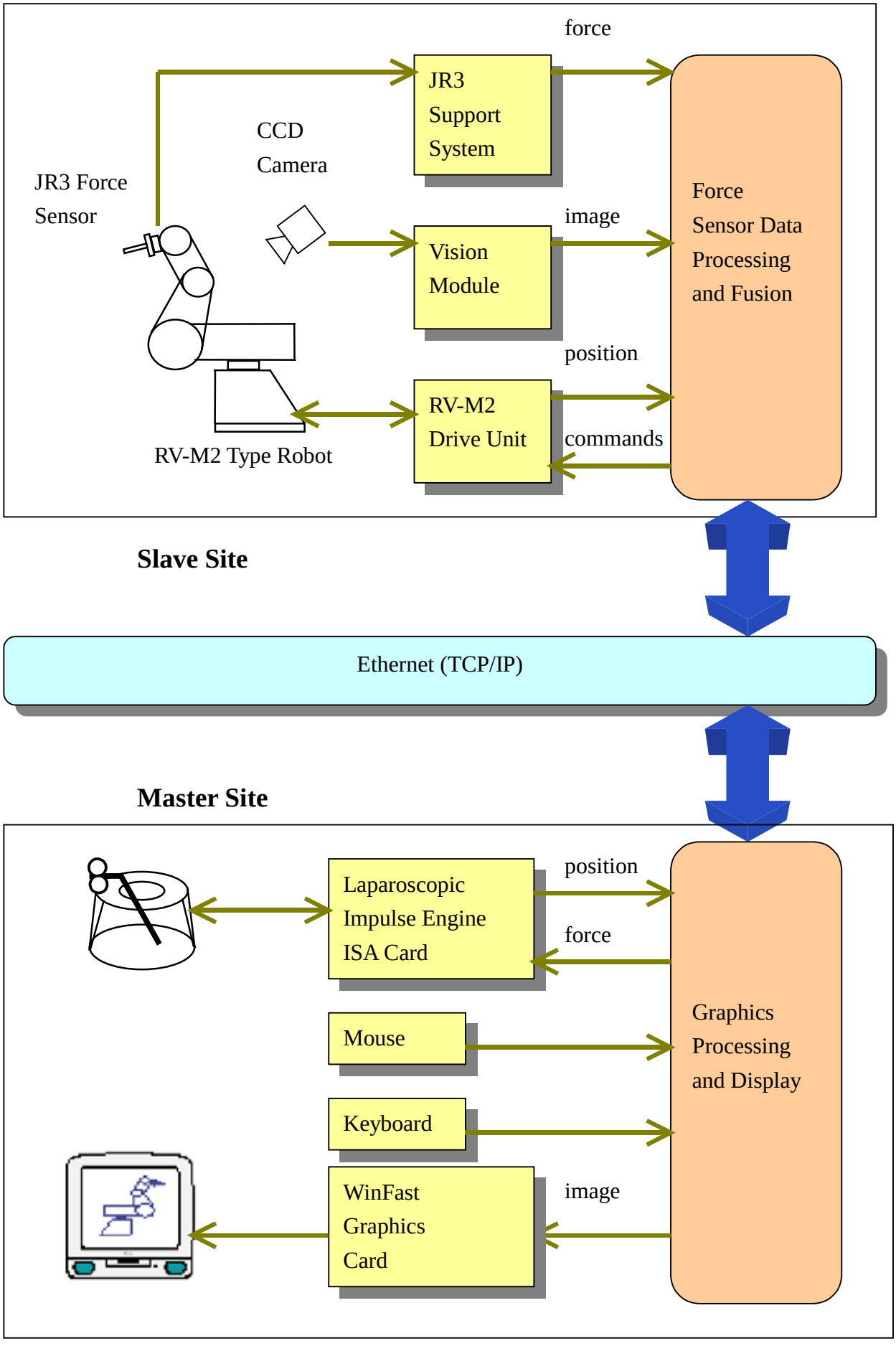


圖 2.2　分散式架構之遙控機器人系統圖

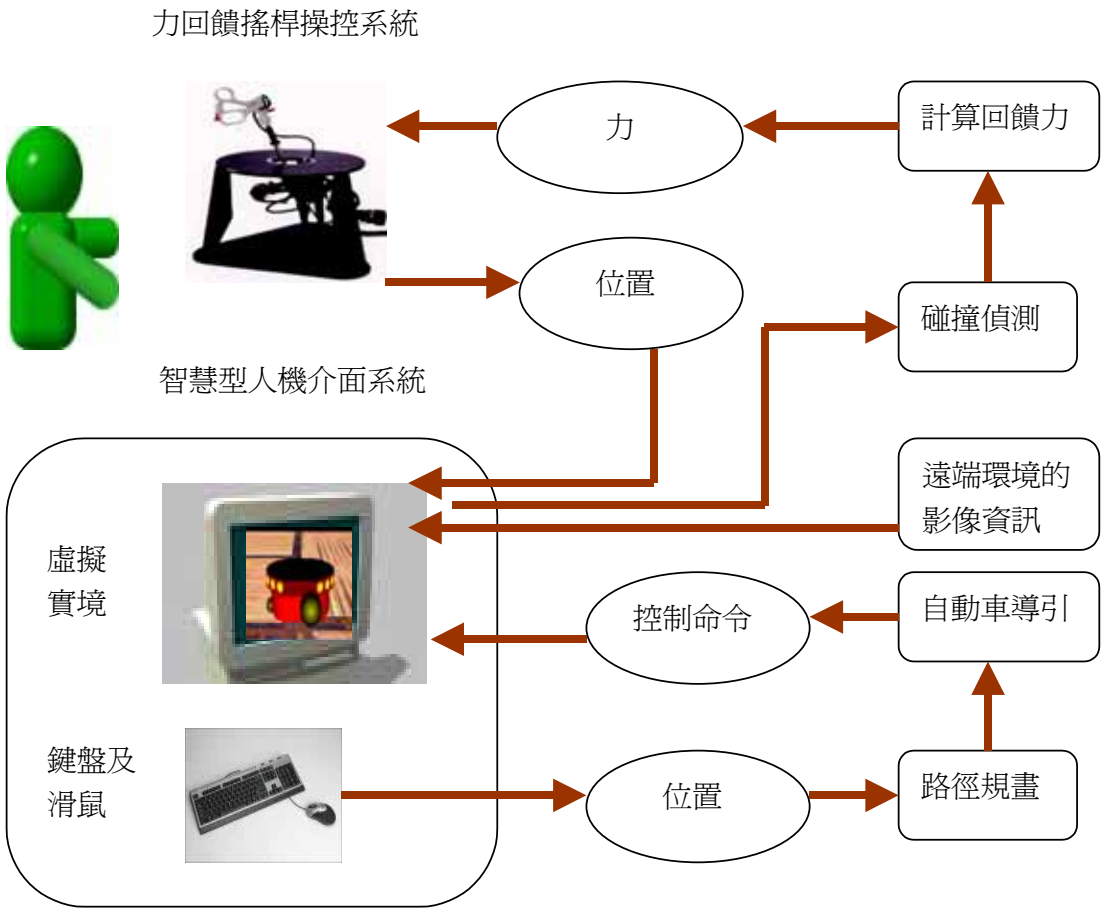


圖 2.3 虛擬實境安全巡邏自動車遠端呈現與操控系統架構圖

2.2 智慧型人機介面系統

爲了讓使用者在操控自動車時，能夠更加輕鬆和方便，我們發展出智慧型人機介面系統。使用者在類似如圖 2.4 的虛擬環境 2D 平面地圖上，可以得知自動車目前所在的位置，同時也可以在此介面上選取目的地，並讓系統自動導引自動

車避開障礙物並到達目的地。要實現這個系統，首先，我們發展出一經過改良的選取方式，讓使用者可以在此介面上選取目的地，然後我們利用 A* 演算法來規畫路徑，產生出避開障礙物的最佳路徑，最後我們利用 Bezier 曲線使路徑平滑化，以下將針對這三個部分來加以介紹。

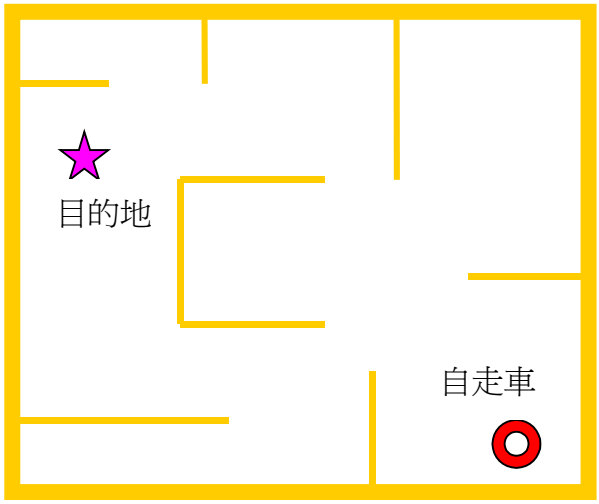


圖 2.4 虛擬環境 2D 平面地圖

2.2.1 改良式選取方法

爲了讓操作者在操控時，能夠更加流暢，在此提出一種改良式選取方法，直接利用游標在虛擬實境裡選取自走車的目的地，省去了用鍵盤輸入目的地的不便。本論文我們使用微軟的視窗系統來顯示虛擬環境，並作爲使用者操控自走車的介面。因爲虛擬世界是 3D 立體的物件，而顯示用的螢幕是 2D 平面的，爲了能在虛擬世界裡選取欲抵達的目的地，我們需要將虛擬世界座標系存在之物件映射到螢幕視窗上的位置算出，當使用者以滑鼠移動游標到目的地並作出選取動作

時，以游標在視窗上的座標位置比對之前算出物件的視窗座標位置，相同者即爲操作者所選取的自走車目的地。再由所接收到的目的地，利用路徑規劃的方法產生避往障礙物的最佳路徑，而虛擬物件映射到視窗界面，如圖 2.5 所示，其過程將由下面介紹。

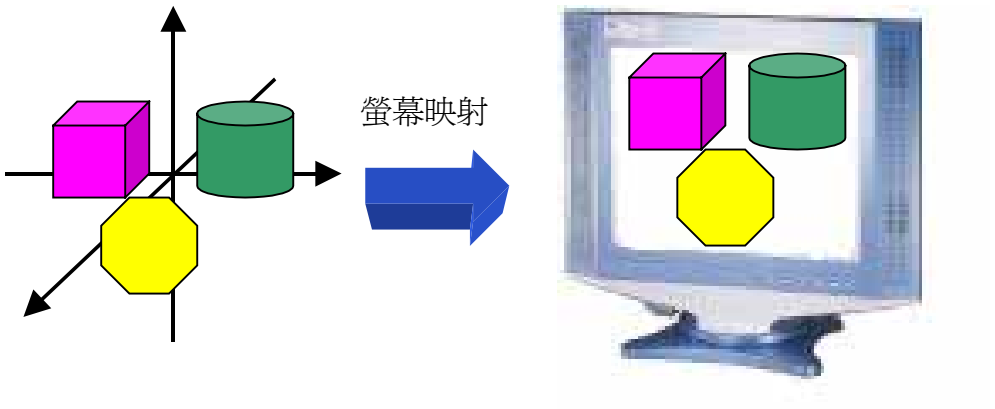


圖 2.5 將虛擬世界座標系中的觀測物體映射到螢幕上

首先介紹虛擬實境是如何呈像在電腦螢幕上，在建構完虛擬世界場景之後，要決定觀測者的位置及方向，並將該方向存在之物件投影至一視平面上，則該視平面上所投影之物件即顯示在螢幕上之視窗內。欲決定觀測者的位置位置及方向，需要三個向量，分別爲眼睛位置、觀測位置、和向上向量，如圖 2.6 所示，眼睛位置代表「觀測者的位置」，而觀測位置代表「觀測物的位置」，知道觀測者及觀測物的位置之後，我們仍需要一個方向向量以區分正反方向，在此以向上向量代表正方向。而 u、v、n 代表觀測座標向量，分別由 n = 眼睛位置 – 觀測位置 、 v = 向上向量 、和 u = v x n 計算出。

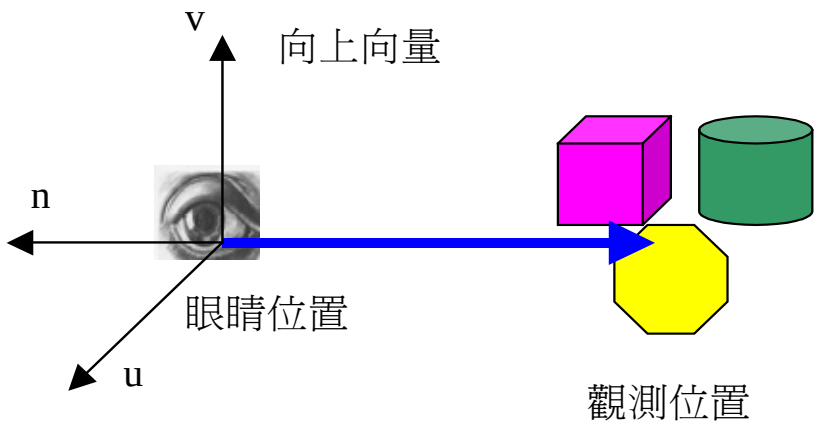


圖 2.6 觀測座標向量

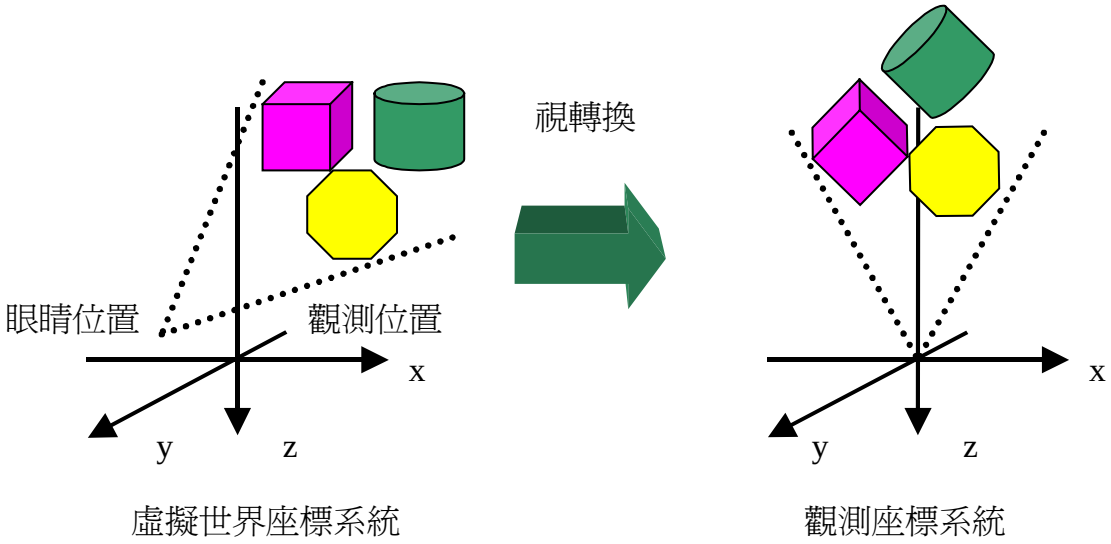


圖 2.7 視轉換示意圖

下面介紹如何計算虛擬世界座標系中物件映射到螢幕視窗上的位置，第一步將虛擬世界座標系統轉換成觀測座標系統，在此稱這種轉換爲視轉換，如圖 2.7 所示。視轉換分爲兩個部份，一個是平移轉換，另一個是旋轉轉換。平移轉換是將座標系統的原點，平移到眼睛位置。旋轉轉換則是將 u、v、和 n 分別映射至座標系中 x 軸、y 軸、和 z 軸的單位向量，也就是 $(1,0,0)$ 、 $(0,1,0)$ 、和 $(0,0,1)$ 。我們令平移轉換矩陣爲 T，旋轉轉換矩陣爲 R， 眼睛位置爲 (e_x,e_y,e_z) ，根據平移轉換矩陣和旋轉轉換矩陣的意義，我們可以得到下列兩式:

$$T\begin{bmatrix} e_x \\ e_y \\ e_z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.1}$$

$$R\begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

根據(2.1) 、 (2.2) 式，我們可以推導出平移轉換矩陣及旋轉轉換矩陣分別爲:

$$T = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$R=\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

最後，我們要將在轉換到觀測座標系統的物件，投影到視平面上，如圖 2.8 所示，我們可以得出下列三式:

$$\frac{x_p}{x}=\frac{z_p}{z} \tag{2.5}$$

$$\frac{y_p}{y}=\frac{z_p}{z} \tag{2.6}$$

$$z_p=-d \tag{2.7}$$

其中，d 代表眼睛座標到視平面的距離。由 (2.5)~(2.7) 式解聯立可得 x_p 、 y_p 如下:

$$x_p=\frac{x\cdot d}{-z} \tag{2.8}$$

$$y_p=\frac{y\cdot d}{-z} \tag{2.9}$$

根據所得出來的 x_p 、 y_p ，我們可以得知物件在螢幕視窗上的相對應位置，再藉由比對游標的位置，進而達成自動車目的地之選取。

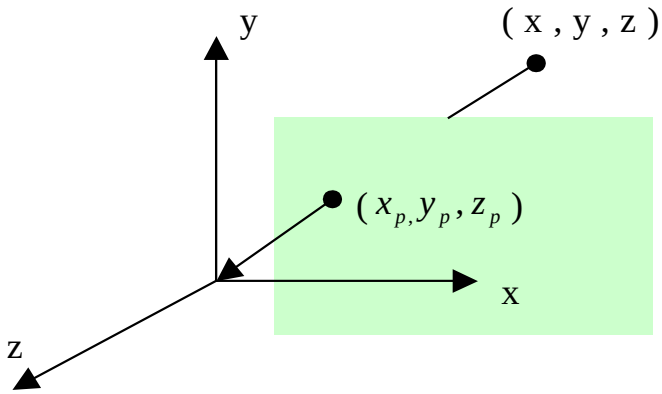


圖 2.8 投影示意圖

2.2.2 路徑規劃

路徑規劃有很多種方法可以達成，其中較爲廣泛應用於自動車之路徑規劃的方法稱爲視圖法(Visibility graph method)[14]。這個方法是基於以下原理：將自動車的所有可移動空間標示成藉由兩兩相連之點所結成的網路，則剩下的問題是搜尋這個網路，找到起始點到終點的最短距離。搜尋的方法大致分爲兩類，一類稱爲全域搜尋(Global planning)，是由詳細地搜尋所有的網路而得出答案之方法；而另一類稱爲區域搜尋(Local planning)，藉由某種探索方法(Heuristic)來導引其搜尋之方向，而得出答案之方法。一較具代表性的全域搜尋法稱爲 Dijkstra's algorithm [5]，它從起始點開始向外搜尋，直到它達到終點爲止。另一較爲有名的區域搜尋法爲 BFS(Best-First-Search) algorithm[19]，它以最接近終點的點作爲搜尋的下一目標點，減少其搜尋之時間。比較 Dijkstra's algorithm 和 BFS algorithm

的搜尋範圍及所找出的路徑，我們可以發現 BFS algorithm 的搜尋範圍通常比 Dijkstra's algorithm 小，這是因爲 BFS algorithm 有利用搜尋方法(Heuristic)，但是 BFS algorithm 無法保證得出最佳路徑。

較廣爲人應用的 A* 演算法[9]，是在 1968 年由 Hart 等三人所提出，只要路徑存在，它可以保證回傳最佳路徑，同時擁有搜尋範圍較小的優點。本論文所發展的智慧型人機介面系統即利用此法，當使用者利用智慧型選取輸入自走車的目的地後，系統就可以用 A* 演算法搜尋格點化之後的地圖，同時將最佳路徑回傳。接下來介紹 A* 演算法，及其所使用的探索函數(Heuristic function)。

A* 演算法沿著從起始點 N_{init} 所產生的路徑(path)，探索路線圖(graph) G 。在每個反覆運算過程的一開始，會有一些演算法已經拜訪(Visited)過的點。對每一個已拜訪過的點 N ，都有一條或數條路徑連接起始點 N_{init} 和 N ，但 A* 演算法只記憶最小成本的路徑[17]。這些路徑都是藉由指標存在樹狀結構 T 中。在 A* 演算法中，每一個點 N 都可以經由成本函數來估算，從起始點 N_{init} 經過點 N ，到終點 N_{goal} 的最小成本路徑。成本函數的計算如下:

$$f(N) = g(N) + h(N) \tag{2.10}$$

其中， $g(N)$ 是 N_{init} 到 N 之間的最小成本， $h(N)$ 是 N_{goal} 到 N 之間所估算的最小成本。A* 演算法的輸入包含了 G 、 N_{init} 、 N_{goal} 、 h 、 k ，而 k 是判斷兩點間成本之函數，所有在 G 中的點，一開始都是未拜訪(Unvisited)的狀態。A* 演算法使用稱爲 $OPEN$ 的名單，藉由 f 的挑選，將 G 中的點儲存於其中。

A* 演算法的演算過程如下：

開始：A*由起始點 N_{init} 開始，通常將 $g(N_{init})$ 設爲 0，並且計算 $f(N_{init})$ ，並將起始點放入 *OPEN* 中。

步驟一：演算法接著從 *OPEN* 中挑 $f(N)$ 最小的點出來，假如 *OPEN* 是空的，則 N_{init} 到 N_{goal} 之間並無任何路徑，演算法結束，假如挑出來的點是 N_{goal} ，則演算法由樹狀結構 T 中重建從 N_{init} 到 N_{goal} 之間的路徑並傳回，演算法結束。

步驟二：若挑出來的點不是 N_{goal} 且未被拜訪過(Unvisited)，則對每個相鄰點，演算法用指標指向原本之點並存入 T 中，並把這些點放入 *OPEN* 中，並且標上已拜訪(Visited)。

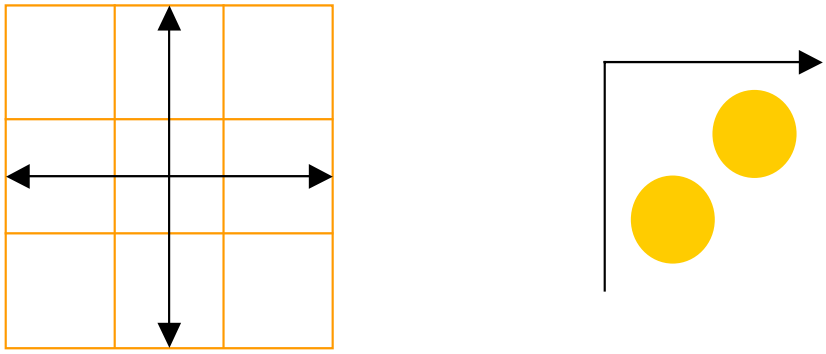
步驟三：若相鄰的點是已拜訪，則演算法跳過此點，但若是此點有比原來的點更佳的路徑，則將原本 T 中的指標由原來的點改到此點，並重新將此點放入 *OPEN* 中。

步驟四：一旦每個相鄰的點都完成上面三個步驟，一開始從 *OPEN* 挑出來的點就改設爲已拜訪過的狀態(Visited)，再從 *OPEN* 中挑一個新的點重覆以上的步驟。

附錄一列出 A* 演算法的文件用程式(Pseudo code)附於附錄一，可供參考。

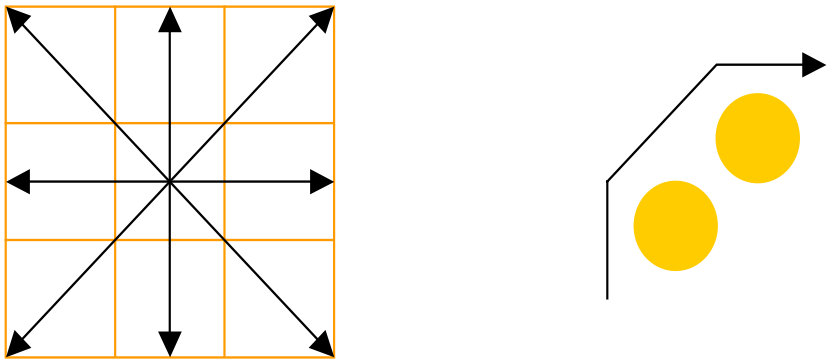
探索函數 $h(N)$ 估算 N_{goal} 到 N 之間的最小成本。探索函數的選擇相當重要，因爲它影響到 A* 演算法的結果。若是我們將 $h(N)$ 直接令爲 0，則 A* 演算法就會變成 Dijkstra 的演算法(Dijkstra's algorithm)[5]。雖然 Dijkstra 的演算法可以保

證永遠找到最佳路徑，但它的效率較差。事實上，若 $h(N)$ 永遠小於或等於 N_{goal} 到 N 之間的最小成本，則 A* 演算法可以保證永遠找到最佳路徑。但是 $h(N)$ 愈小，A* 演算法所需搜尋的點愈多，所以效率愈差。



(a) 搜尋方向　　(b) 路徑

圖 2.9 探索函數使用 Manhattan distance 的 (a) 搜尋方向 及 (b)路徑



(a) 搜尋方向　　(b) 路徑

圖 2.10 探索函數使用 Euclidean distance 的 (a) 搜尋方向 及 (b)路徑

一個基本的探索函數是曼哈頓距離(Manhattan distance)，它的計算式如下所示:

$$h(N) = \left|N \bullet x - N_{goal} \bullet x\right| + \left|N \bullet y - N_{goal} \bullet y\right| \quad (2.11)$$

如果我們使用曼哈頓距離(Manhattan distance)作爲探索函數，代表了搜尋方向有東、南、西、北，四個方向，如圖 2.9 所示。若是將搜尋方向增加到八個，則所得出的路徑會較爲平順，如圖 2.10，但是探索函數就必須要使用歐幾里得距離(Euclidean distance)，計算式如下:

$$h(N) = \sqrt{(N \bullet x - N_{goal} \bullet x)^2 + (N \bullet y - N_{goal} \bullet y)^2} \quad (2.12)$$

2.2.3 路徑平滑化

我們利用 A* 演算法所做的路徑規畫，雖然已找到地形上之最佳路徑，但該路徑由格點所組成，並非一平滑的曲線。要解決這個問題，我們可以利用 Bezier 曲線，完成所規畫的路徑平滑化之目標，Bezier 演算法早在 1960 年由一名汽車工程師 Pierre Bezier 所提出的，剛開始被運用在輔助設計汽車工業裡，後來此方法就被廣泛的討論[1,2]。

真實世界裡由於存在著各式各樣的圖案，所以若想利用電腦繪圖來製作出形狀外觀曲線，其幾乎都利用片段曲線連接產生出來的，而無法利用特定的方程式來描述它，因爲特定的方程式所能產生的曲線形狀是非常有限的，而 Bezier 曲線就是採用片段連接曲線的方式來做，使其可描述出各種曲線型式。曲線產生的方式可分爲兩種，一種是內插 (Interpolation) 的方法，其所產生的曲線會通過空間中所有的控制點座標，另一種是近似 (Approximation) 的方法，其所產生的的曲線只會通過曲線上第一個和最後一個控制點座標而已，以此種方式所產生的曲

線形狀，是由控制點所連接成的多邊形來決定的，因此只要改變控制點座標，就可以繪製出產生出一個近似所想要的曲線。

現假設有$n+1$個控制點在 3 維空間中，其 Bezier 參數曲線方程式表示如下，下式也稱之爲n階的 Bezier 曲線方程式：

$$C(u)=\sum_{i=0}^{n}B_{i,n}(u)P_i \qquad 0\le u\le 1 \tag{2.13}$$

其中n是多項是的次數，$\{P_i\}$表示其$n+1$個控制點，而$\{B_{i,n}(u)\}$爲混合函數(blending function)，其可描述成：

$$B_{i,n}(u)=\frac{n!}{i!(n-i)!}u^i(1-u)^{n-i} \tag{2.14}$$

其中i是從 0 到n，表示第幾個控制點。在(2.13)式中，尚需滿足$u\in[0,1]$這個條件，即當$u=0$時，則混合函數只有$B_{0,n}(0)$爲 1，其餘都爲 0，當$u=1$時，則混合函數只有$B_{n,n}(1)$爲 1，其餘都爲 0，因爲這樣所以才會使得 Bezier 曲線只會通過第一個控制點和最後一個控制點的這特性，另外混合函數還有一個特性就是：

$$\sum_{i=0}^{n}B_{i,n}(u)=1 \qquad 0\le u\le 1 \tag{2.15}$$

圖 2.11 (a)-(c) 爲$n=1,2,3$的三次多項式混合函數$\{B_{i,n}(u)\}$的圖形，根據(2.14)式可求得表示式爲：

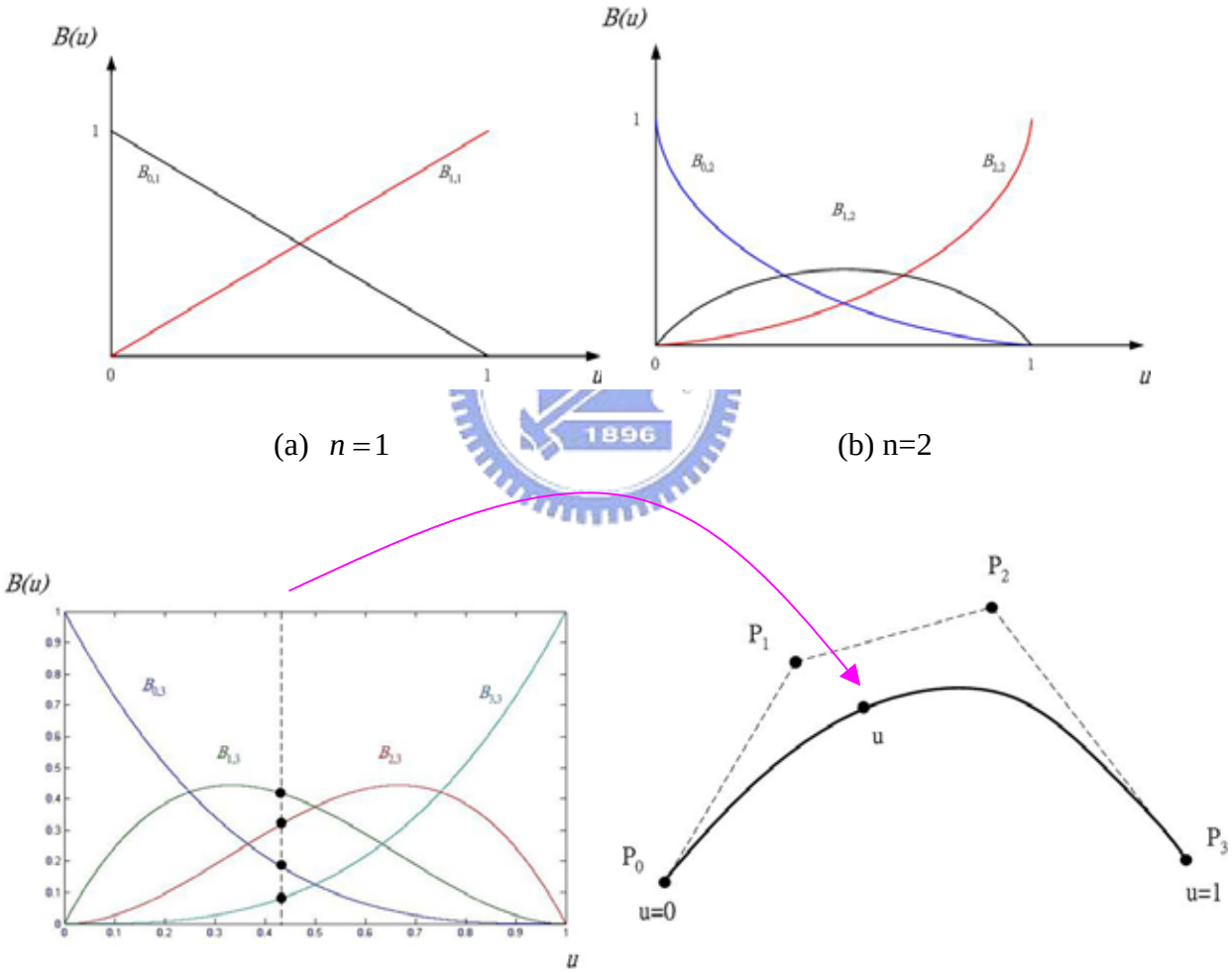
$$\begin{aligned}
B_{0,3}(u)&=(1-u)^3\\
B_{1,3}(u)&=3u(1-u)^2\\
B_{2,3}(u)&=3u^2(1-u)\\
B_{3,3}(u)&=u^3
\end{aligned} \tag{2.16}$$

由(2.16)式可知 $B_{i,3}(u)$ 只和曲線參數 u 有關，如圖2.13(c)所示。接著利用(2.13)式來計算Bezier曲線，表示如下：

$$C(u)=\sum_{i=0}^{3}B_{i,3}(u)P_i$$

$$=P_0B_{0,3}(u)+P_1B_{1,3}(u)+P_2B_{2,3}(u)+P_3B_{3,3}(u)$$

$$=P_0(1-u)^3+P_1 3u(1-u)^2+P_2 3u^2(1-u)+P_3u^3 \qquad (2.17)$$



(a) $n=1$ (b) n=2

(c) n=3 (d) Bezier曲線範例

圖2.11 (a) n=1之Bezier混合函數, (b) n=2之Bezier混合函數,

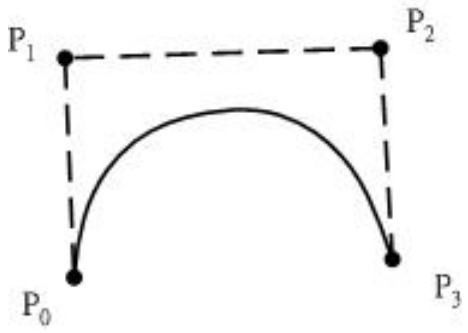
(c) n=3之Bezier混合函數,和(d)n=3之Bezier曲線範例

圖2.11(d)爲n=3的Bezier曲線範例，其中 P_0 、 P_1 、 P_2 和 P_3 爲控制點，由(2.17)式可求得任意 u 值相對於Bezier曲線上的位置，如圖中在曲線上的 u 點位置座標，它是由其相對應的 u 值經由混合函數計算得到其加權和值 (Weighted sum)，也就是每個控制點分別給予此 u 點一個加權值，在帶入Bezier曲線方程式即可得到相對於空間中的座標點。

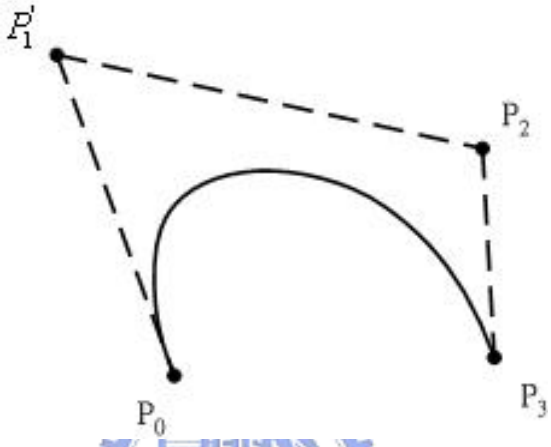
在圖2.12中舉出一例子，其可藉由更改控制點在空間中的位置，即可以改變Bezier曲線的形狀，從圖中觀察可知，當我們將控制點 P_1 位移到其他座標點上，曲線就會隨著 P_1 的變化，產生一近似這些控制點的曲線形狀。下面列出幾個Bezier曲線的重要特性[21]：

- 曲線的形狀會近似由控制點所組成的控制多邊形(Control polygon) ；
- 曲線的起始點爲 P_0 ，終點爲 P_3 ，也就是 $P_0 = C(0)$ 且 $P_3 = C(1)$ ，另外曲線不會通過 P_1 和 P_2 這兩點；
- 曲線起點的切線方線和 P_0 到 P_1 的方向是一樣的，相同地，曲線終點的切線方向和 P_2 到 P_3 的方向是相同的；
- 根據控制點的位置，曲線會包含在凸多邊形內；
- 如果 P_0 、 P_1 、 P_2 和 P_3 都在同一直線上的話，則所產生出來的曲線也會是一條直線。

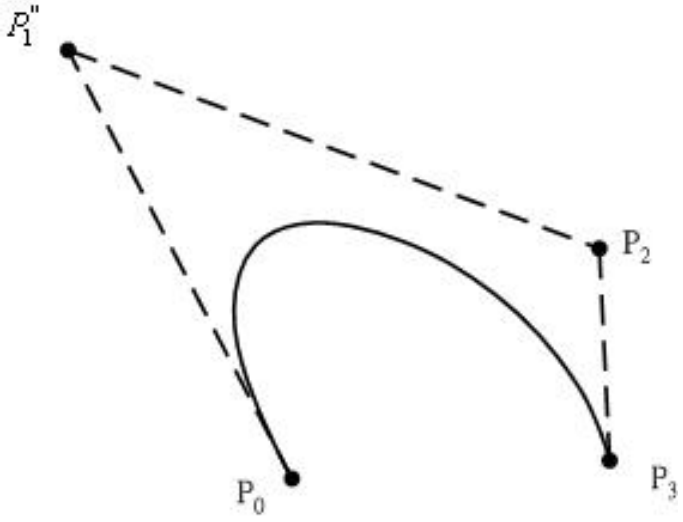
由上面幾個特性可知，曲線的控制是以一個很直覺式的方式去控制產生曲線的，也就是只要改變控制多邊形上的點座標，就可以達到曲線控制目的。



(a) 未變化前的Bezier曲線



(b)將 P_1 移到 P_1' 位置



(c) 將 P_1 移到 P_1'' 位置

圖 2.12 控制點對 Bezier 曲線的影響：(a)未變化前的 Bezier 曲線,(b)將 P_1 移到 P_1' 位置,和(c)將 P_1 移到 P_1'' 位置

2.3 力迴饋搖桿操控系統

對遠端操控的使用者而言，除了視覺資訊的提供外，力覺(Force reflection)資訊的呈現也相當重要[29]，將力回饋加入操控系統中，可以有效地增加整個系統的操控能力，例如當自動車接近障礙物時，遠端的使用者如果只有視覺上的回饋，很容易因爲視覺上的誤判，讓自動車和障礙物產生碰撞，這時候如果讓使用者在判斷障礙物距離遠近時，多一種力覺上的資訊，則發生碰撞的機率將小很多，而使用者在操控上的效率也會提高，在 1988 年，Pique 等三人就說明了力回饋能有效地增加操控的穩定性及放置物體的工作效率 [20]。在本論文中，我們採用一種平面與探針(Plane and probe)方法之概念，讓使用者操控自動車在接近障礙物時，能從搖桿傳回的力資訊中，幫助使用者判斷障礙物之距離與方向，並且能較有效率地避開障礙物。

人類在力覺上對力資訊的更新頻率要求比視覺資訊高很多[4]，在實現力回饋搖桿操控系統時，特別將力資訊的處理流程加以簡化，平面與探針方法的概念最早是由 Adachi[2]等人在 1995 年提出，目的是讓操控者在使用力覺裝置和虛擬實境中物體作接觸時，有較高頻率的力回饋，原理是在碰撞偵測時，將虛擬環境中的物體視作爲一個虛擬平面，如此一來，將可減少系統在處理碰撞偵測之計算上所耗費的時間，並且加快力資訊的處理效率。。而平面與探針(Plane and probe)這個方法最早出現在 1996 年[15]。

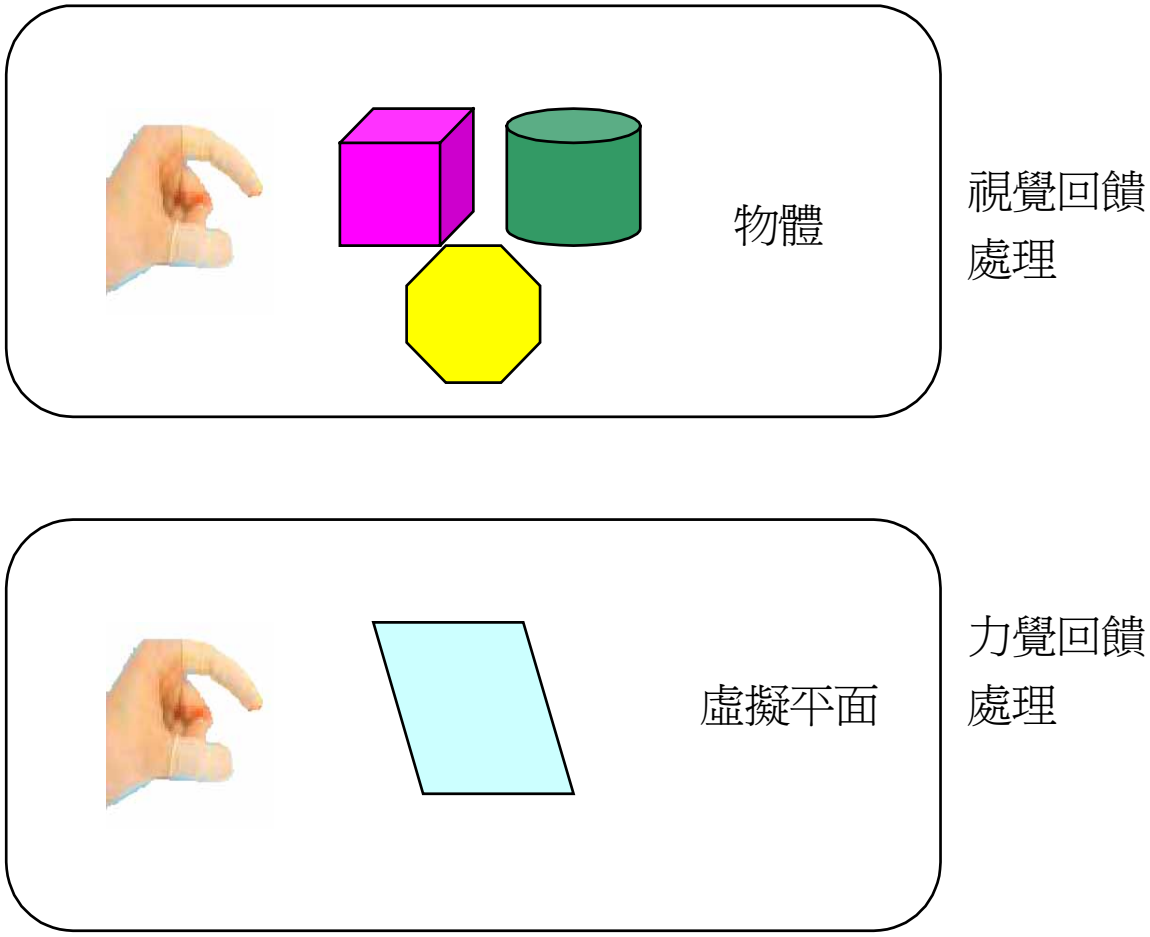


圖 2.13 平面與探針方法中視覺及力覺回饋處理之示意圖

如圖 2.13 所示，我們將虛擬環境中的物體視作爲一個虛擬平面(Virtual plane)，在使用者的視覺回饋上是原本的物體，但在處理力覺回饋時，就利用虛擬平面做碰撞偵測並計算回饋力，並且將手指視作爲一個虛擬探針(Probe)，如圖 2.14，在計算回饋力時，將此虛擬平面可以用下面的式子代表:

$$ax + by + cz + d = 0 \qquad (2.18)$$

其中 x, y, z 代表虛擬環境的世界座標。如此一來，碰撞偵測的判斷只需將代表機器手臂的虛擬探針，其尖端的點代入 2.18 式即可，將大大地增加其效率。而虛擬探針插入虛擬平面的深度(Δr)，則根據下式估算得出:

$$\Delta r = \left| ax_0 + by_0 + cz_0 + d \right| / \sqrt{a^2 + b^2 + c^2} \tag{2.19}$$

其中 x_0, y_0, z_0 代表虛擬探針的尖端位置。由(2.19)式，我們可以得知回傳力 F_v 的大小，如下式:

$$F_v = K \bullet \Delta r \tag{2.20}$$

其中， K 爲虛擬物體的彈性係數。

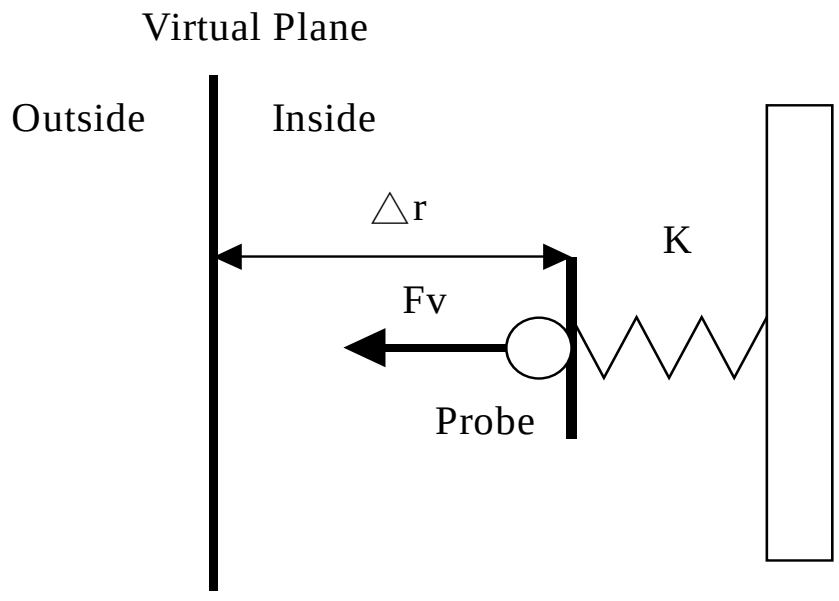


圖 2.14 虛擬力產生之示意圖

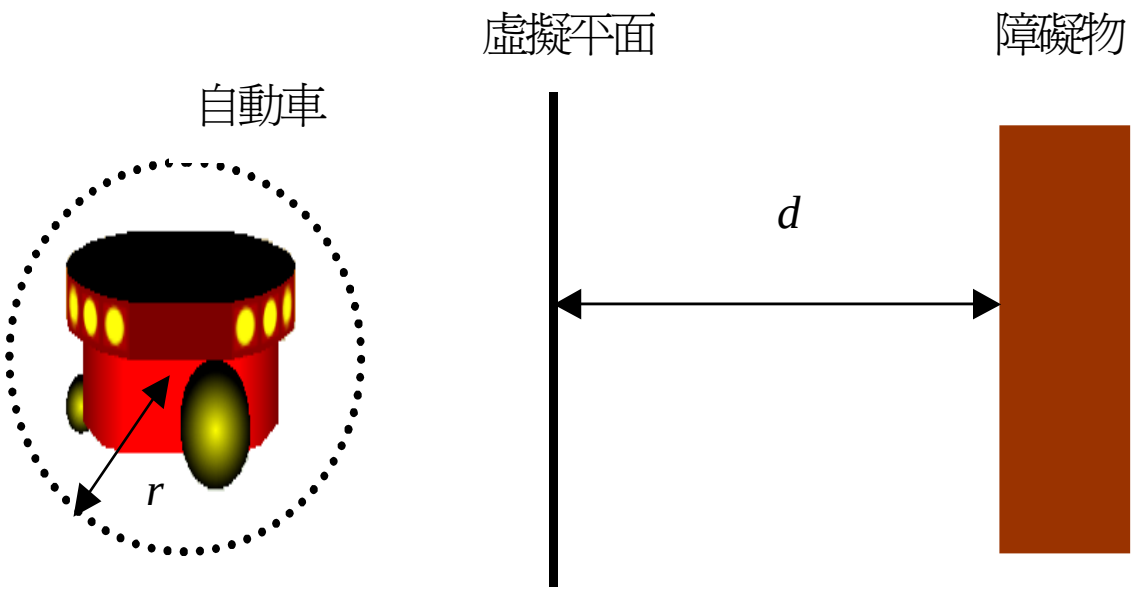


圖 2.15 自動車、障礙物及虛擬平面間之關係示意圖

我們在力回饋搖桿操控系統中，引進如上之方法，如圖 2.15 所示，首先取一個半徑爲 r 的圓，其面積需涵蓋整個自動車，利用其圓點當作平面與探針方法中的探針。接著在障礙物前面設計一道虛擬平面，作爲計算虛擬力的來源。因爲不希望自動車和障礙物發生碰撞，所以虛擬平面和障礙物之間的距離 d 設爲 r 的 2 到 3 倍。當代表探針的圓點插入虛擬平面的時候，代表自動車和障礙物之間的距離已經過於接近。這時候，使用者會開始接受到由力回饋搖桿傳回來的警告力，自動車愈接近障礙物，代表探針插入平面的深度也愈深，計算出來的虛擬力也就愈大。使用者可以藉由力回饋搖桿所傳回來的警告力變大，得知所操控的自動車離障礙物愈來愈近，並且由力的方向來判斷障礙物的方向。同樣的，當使用者迴避障礙物之後，力回饋搖桿所傳回來的警告力也會變小，而超過一定距離(自動車脫離虛擬平面)之後，自動車和障礙物已無發生碰撞的危險，警告力也就消失。